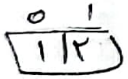
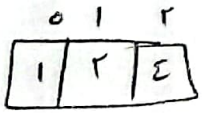
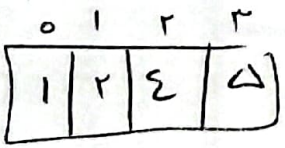
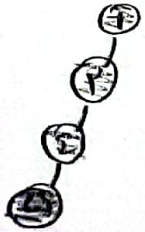
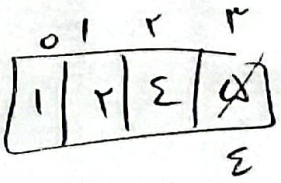
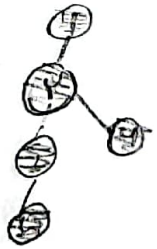
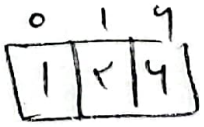
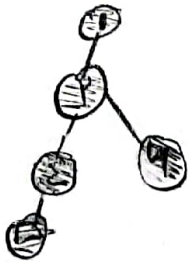
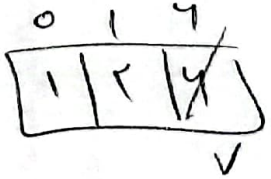
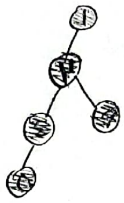
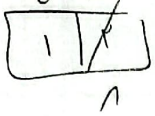
① هر راس را هنگام خارج شدن از همان starting_time وارد یک استک می‌کنیم و هنگام مشکی شدن از همان finishing_time از استک پاک می‌کنیم.

time	استک: starting time / finishing time
۰	
۱	 
۲	 
۳	 
۴	 
۵	 
۶	 
۷	 

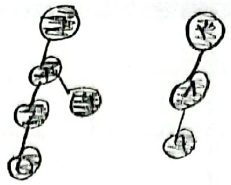
time

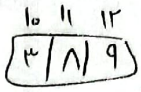




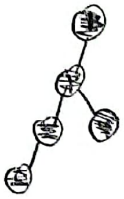
۱

time

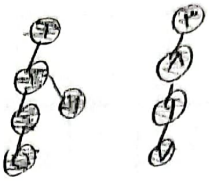


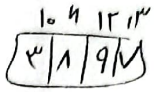


۱۲

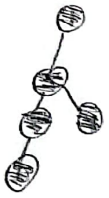


۹

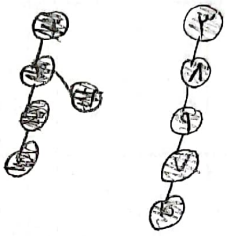


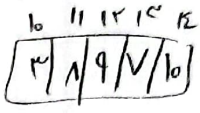


۱۳

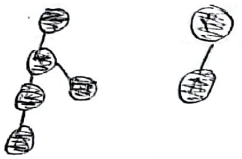


۱۰





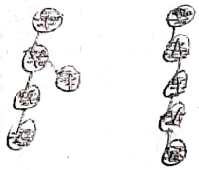
۱۴



۱۱

حل ۵ هم، راس ها پیمایش شده راس های خاکستری را از استک خارج کرده و مشکی می کنیم

داریم:



time

۱۹

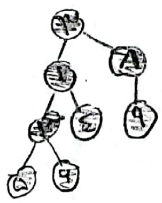
back_edge : ۷→۳ ۱۰→۹

forward_edge: ۱→۴ ۲→۵ ۹→۱۰

cross_edge: ۹→۵ ۷→۶ ۳→۱ ۱۰→۱

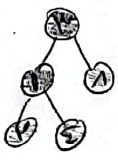
پیمایش bfs:

۳

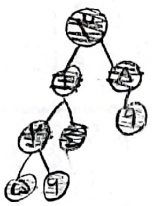


۳,۸,۱,۶,۵,۹,۴

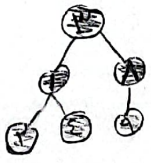
۳,۸,۱



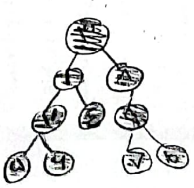
۳,۸,۱,۲,۴



۳,۸,۱,۶,۵,۹,۴

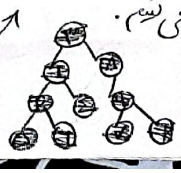


۳,۸,۱,۲,۴,۹



۳,۸,۱,۶,۵,۹,۴,۷,۱۰

حل ۶ هم، راس پیمایش شده اند راس های باقی مانده یعنی ۱۰،۷،۹،۵،۱ را به ترتیب خارج می کنیم.



(۲) ابتدا پیمایش dfs را انجام داده و درخت dfs را می‌سازیم. در ابتدا یک راس تصادفی را انتخاب و وضعیت آن را به عنوان ملاقات نشده علامت گذاری می‌کنیم. مرحله بعدی محاسبه عمق راس انتخاب شده است عمق هر راس ترتیب بازدید آنهاست.

visited

Depth

سپس باید کمترین عدد کشف را محاسبه کرد که برابر است با: عمق راس قابل دستیابی از هر راس با در نظر گرفتن back-edge

low(v)

در درخت dfs.

بعد از محاسبه این دو مقدار برای اولین راس انتخاب شده الگوریتم سپس رئوس مجاور آن را جستجو می‌کند. بررسی می‌کند که رئوس مجاور آن قبلا ملاقات شده اند یا نه اگر نشده بودند به عنوان راس فعلی depth و low برای آن محاسبه می‌شود.

بعد از محاسبه این دو مقدار برای تمامی رئوس، الگوریتم‌ها یک جفت رئوس را می‌گیرند و بررسی می‌کنند راس برشی هست یا خیر.

فرض کنیم دو راس v_1 و v_2 باشند و در نظر می‌گیریم v_1 یک راس مادر و v_2 راس فرزند است اگر $low(v_2)$ بزرگتر یا مساوی $depth(v_1)$ باشد آنگاه v_1 راس برشی است.

یک مورد خاص وجود دارد که ریشه راس برشی است اگر و فقط اگر بیش از یک فرزند داشته باشد.

شبه کد سوال (۲)

```
void AP (s,d)
  visited[s] = TRUE;
  depth[s] = d;
  low[s] = d;
  for each k ∈ Adj(s) {
     if (visited[s] == FAISE)
          AP(k, d+1);
     low[s] = min(low[s], low[k]);
     if (low[k] ≥ depth[s]) {
          if(s = not root node  or  number of children(s) > 1 )
               print("s is an articulation point")
     }
  }
```

③ از ریشه شروع می‌کنیم پیمایش bfs را انجام می‌دهیم و فاصله هر راس از ریشه را ذخیره می‌کنیم و بار دیگر از u شروع می‌کنیم و پیمایش bfs را انجام می‌دهیم فاصله هر راس از u را ذخیره می‌کنیم پیچیدگی برابر $O(2n) = O(n)$ است. پس از راسی بیش‌ترین فاصله را از ریشه دارد شروع می‌کنیم. اگر فاصله از ریشه بیشتر از فاصله از u بود آنجا راس مقصد است و جواب برابر با فاصله از ریشه می‌باشد اگر بیشتر نبود به سراغ راس‌های دیگر آن سطح و پس سطوح بالا می‌رویم پیچیدگی در اینجا نیز $O(n)$ است پس در کل پیچیدگی $O(n)$ می‌باشد.

④ از راس s شروع می‌کنیم و پیمایش bfs را انجام می‌دهیم و فاصله هر راس از s را حساب می‌کنیم. یک پیمایش bfs از t نیز انجام داده و فاصله هر راس از t را هم پیدا می‌کنیم. هنگام اضافه کردن یال بین دو راس باید آن یال بین d_s و d_t قرار گیرد تا فاصله کاهش پیدا کند. بنابراین اگر فاصله دو سر یال اضافه شده به اضافه یک (خود یال) از فاصله اولیه d_s و d_t کمتر نبود آن یال را جمع جزو جواب در نظر می‌گیریم به این ترتیب برای یال‌های باقی مانده k از اردر n^2 است. برای هرکدام با $O(n)$ متوجه می‌شویم جزو جواب است یا خیر.

(۵) از یک ایستگاه دلخواه پیمایش DFS را شروع کرده در هرجا که back_edge داشتیم راس های ابتدا و انتهای آن و نیز راس‌های بین ابتدا و انتهای back_edge روس درون دور هستند و اختلاف آنها با دور صفر می باشد. پیچیدگی تا این قسمت برابر $O(n+m)$ یا $O(n)$ می باشد حال از دو سر back_edge پیمایش رفت را شروع می‌کنیم و برابر فاصله آن راس تا دور می گذاریم پیچیدگی $O(n)$ است پس در کل پیچیدگی برابر $O(n)$ می باشد.

(۶) نکته: مولفه‌های قویا همبند یک گراف جهت دار با مولفه‌های قویا همبند معکوس آن گراف جهت دار یکی است.

الگوریتم کسارایو: نکته بالا پایه اصلی این الگوریتم است. به این ترتیب که ابتدا یک استک خالی را در نظر می‌گیریم، گراف جهت دار داده شده را G در نظر می‌گیریم این گراف را پیمایش می‌کنیم پیمایش این گراف را از یک راس دلخواه به وسیله Dfs انجام می‌دهیم و وقتی Dfs کار خودرا با یک راس به اتمام رساند آن را داخل استک قرار می‌دهیم به همین ترتیب ادامه می‌دهیم تا کل گراف پیمایش شود. در این مرحله تمامی راس‌ها در داخل استک قرار دارند.

سپس گراف معکوس G یعنی G^T را به دست می‌آوریم. این بار گراف معکوس را با توجه به اولویت راس‌ها در استک پیمایش می‌کنیم به این ترتیب که در ابتدا از راسی شروع می‌کنیم که در بالای استک قرار دارد. فرض کنید راس v در بالای استک است Dfs را از آن شروع می‌کنیم فرض کنید درانتها نام w به پایان برسد. تمام راس‌هایی که در این بین ملاقات شده جزو یک مولفه‌های قویا همبند هستند تمام راس‌های ملاقات شده را از استک خارج و همین الگوریتم را بر روی راسی که در بالای استک قرار دارد تکرار کرده تا زمانی که استک خالی شود.